Une sémantique d'ESTEREL par ensembles partiellement ordonnés de signaux

Guillaume Doumenc

INRIA

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.(1) 39 63 55 11

Rapports de Recherche

N° 1161

Programme 1
Programmation, Calcul Symbolique
et Intelligence Artificielle

UNE SEMANTIQUE D'ESTEREL PAR ENSEMBLES PARTIELLEMENT ORDONNES DE SIGNAUX

Guillaume DOUMENC

Février 1990

RR-1161

Une sémantique d'ESTEREL par ensembles partiellement ordonnés de signaux
A poset semantics for the ESTEREL language

Guillaume Doumenc

I.N.R.I.A. Sophia Antipolis
Route des Lucioles
06 560 Valbonne FRANCE
gdo@mirsa.inria.fr

Résumé. On décrit une nouvelle sémantique formelle du langage ESTEREL. Cette sémantique est donnée sous forme d'ensembles de signaux partiellement ordonnés entre eux. Cette sémantique est définie à partir d'un calcul de processus inhérent à ESTEREL et est ensuite étendue à ESTEREL tout entier. Cette sémantique est directe dans le sens où elle ne fait pas appel à la notion de critère d'observation. elle vérifie le déterminisme des programmes et admet la modularité.
Mots clés : ESTEREL. modularité. ensemble partiellement ordonné.

Abstract. This paper describes a new formal semantics for the ESTEREL language. This semantics is given by sets of ordered signals and is direct in the sense that it does not use abstract criterion. We define for that a processus calculus based on ESTEREL and define this semantics on it. Then we extend this semantics to whole ESTEREL and verify that it preserves determinism and modularity.
Keywords : ESTEREL. modularity. poset.

1 Introduction

Le langage ESTEREL a été développé par G.BERRY à l'Ecole des Mines, pour décrire des systèmes réactifs. Nous présentons dans ce rapport une version calcul de processus de ce langage dont nous donnerons une sémantique opérationnelle directe. Le domaine sémantique que nous considèrerons est la théorie des ensembles. On associe à chaque programme ESTEREL l'ensemble des signaux qui peuvent etre présents dans une réaction. La sémantique que nous définissons est directe dans le sens où elle nous permet de ne plus quotienter la sémantique d'exécution. Comme conséquence, cette sémantique sera modulaire.

Nous montrerons alors que la notion de déterminisme s'exprime par une relation d'ordre entre les signaux. Cette relation représente la notion de cycle de causalité qui si elle ne gène pas pour la compréhension de la sémantique d'un programme, peut parfois amener à un comportement non-déterministe.

Pour cela nous définirons un sous-calcul de processus inhérent à ESTEREL que nous appellerons ESTEREL$^-$ et sur lequel nous définirons notre sémantique. Puis nous montrerons comment cette sémantique peut être étendue à ESTEREL tout entier et nous donnerons quelques exemples de programme avec leur sémantique associée. Pour cela nous utiliserons un outil écrit en Prolog [5].

2 Définition du langage ESTEREL$^-$

2.1 Présentation générale du langage ESTEREL

Le langage ESTEREL est un langage permettant la programmation de systèmes réactifs déterministes ([7]). Chaque stimuli ou réaction des systèmes est représenté par l'occurrence d'un signal :

- un signal peut avoir été émis par l'extérieur : cela représente un stimulus pour le système (top d'horloge, pression d'un bouton, ...).
- un signal peut être émis par le système : cela permet d'agir sur son environnement (commande de moteur, réaction, ...),
- un signal peut être localement présent : cela permet au système d'enclencher des réactions internes; ces réactions se faisant dans l'instant (programmation par *dialogues instantanés*).

Chaque signal est diffusé, contrairement à des signaux de rendez-vous comme en CCS([10]), LOTOS([6]) ou MEIJE([1]).

L'objectif visé est de pouvoir traiter de façon uniforme tout signal comme engendrant sa propre forme de temps, dans le sens où toute réaction introduit une notion d'évènement et donc indirectement d'unité de temps. Pour cela, il convient de négliger le temps d'exécution des instructions internes afin de ne pas

provoquer des perturbations ou des décalages par rapport à ces signaux: c'est l'hypothèse de *fort synchronisme* d'ESTEREL.

ESTEREL est un langage synchrone de haut-niveau, qui permet d'associer à chaque programme un automate fini déterministe qui correspond au squelette de son contrôle et dont l'efficacité d'exécution est maximale. A chaque transition de cet automate correspond une ou plusieures instructions élémentaires de la machine sur laquelle le programme ESTEREL s'exécute. La compilation d'un programme ESTEREL consiste à construire cet automate. Une notion est très importante pour les programmes ESTEREL : *la terminaison*.

Cette notion permet de discrétiser les réactions d'un programme ESTEREL. L'hypothèse de fort synchronisme d'ESTEREL stipule que les délais de réaction sont infiniment rapides. La réaction doit donc terminer suffisamment rapidement pour valider cette hypothèse. On doit réfuter les programmes qui ne *terminent* jamais (ex.: `loop emit` *s* `end`, qui émet indéfiniment le signal `s` et ne termine jamais).

La difficulté de la compilation d'un programme ESTEREL consiste à déterminer la validité de ce programme et à en déduire un squelette optimal (qui n'est pas nécessairement unique). Cette complexité est accrue si on permet de compiler indépendamment différents modules ESTEREL.

2.2 Définition du langage ESTEREL$^-$

On appelle ESTEREL en signaux purs, le langage ESTEREL démuni de variables et de valeurs sur les signaux. Ce langage permet de définir uniquement les structures de contrôle d'un programme, tout en limitant les flux d'informations. Ainsi, seules les synchronisations et les réactions peuvent être décrites. Nous définissons un sous-langage de ESTEREL en signaux purs, que nous appelerons ESTEREL$^-$, qui est une simplification syntaxique de ESTEREL en signaux purs. Notons toutefois, qu'à la compilation, lors de la création de l'automate associé à un programme ESTEREL, seules les informations présentes dans cette forme "calcul de processus" (c'est à dire allégée des valeurs) seront prises en compte.

2.2.1 Signature

La syntaxe concrète du langage est proche de celle de ESTEREL tout entier. Nous définissons le langage ESTEREL$^-$ à partir de la signature suivante :

sortes

SIGNAL.	(type des signaux contenus dans le module)
STAT.	(type des configurations de programmes)
INT.	(type des entiers)
COND	(booléen validant le choix)
IDENT	(identificateur)

REN (renommage syntaxique)

opérateurs

dec-input	:: $(SIGNAL \times STAT)$	— $STAT$
dec-output	:: $(SIGNAL \times STAT)$	— $STAT$
dec-local	:: $(SIGNAL \times STAT)$	— $STAT$
nothing	::	— $STAT$
halt	::	— $STAT$
emit	:: $(SIGNAL)$	— $STAT$
sequence	:: $(STAT \times STAT)$	— $STAT$
loop	:: $(STAT)$	— $STAT$
present	:: $(SIGNAL \times STAT \times STAT)$	— $STAT$
watching	:: $(STAT \times SIGNAL)$	— $STAT$
parallel	:: $(STAT \times STAT)$	— $STAT$
if	:: $(COND \times STAT)$	— $STAT$
trap	:: $(STAT)$	— $STAT$
exit	:: (INT)	— $STAT$
module	:: $(IDENT \times STAT)$	— $STAT$
copymodule	:: $(IDENT \times STAT \times REN)$	— $STAT$

Les deux principales différences avec la syntaxe de ESTEREL en signaux purs sont : les opérateurs de déclarations de signaux (dec-input. dec-ouput et dec-local) et ceux déchappement (trap et exit).

- en ESTEREL$^-$ toute déclaration devient une instruction. Cela permet de simplifier l'énoncé de la traduction mais ne doit pas servir à définir des signaux externes locaux à un bloc d'instructions. De plus on considèrera que les signaux de tous les programmes ont bien été déclarés dans chaque module avant d'être référencés.

- les mécanismes d'échappements sont définis par leur niveau d'imbrication. Ceux-ci étant déduits d'une passe statique sur des échappements nommés.

L'opérateur de choix (if) permet de considérer des conditions qui sont non interprétées. On représente ces conditions par des signaux dont les noms désignent les conditions réalisées.

2.2.2 Module et instruction

Un programme ESTEREL$^-$ est une entité complète et fermée que l'on apelle *module* qui est destinée à etre interfacée avec un environnement extérieur. Un module est lui-meme formé de briques de base que sont les *instructions*. Ces deux notions ne sont pas simplement différenciées pour des problèmes de déclarations syntaxiques mais pour des raisons plus fondamentales. comme la terminaison et les niveaux d'échappement. Un module est la seule entitée pouvant

être importée dans un programme ESTEREL$^-$, et l'opération de compilation ne peut s'appliquer qu'à un module.

3 Une sémantique de ESTEREL$^-$

De nombreuses sémantiques du langage ESTEREL ont été définies ([2], [12], [3], [4]). Elles ont été définies pour représenter et exprimer le plus fidèlement possible la notion de synchronisme fort. Nous nous interesserons surtout à celle de Gonthier [4] et de Boussinot [3]. Pour décrire le langage ESTEREL$^-$, nous utiliserons une sémantique proche de la sémantique comportementale de [4]. Cette sémantique permet de décrire le comportement d'un programme ESTEREL$^-$ en fonction d'évènements extérieurs. Ces deux sémantiques sont basées sur la notion de règles de déduction structurelle ([11],[9]) et donc considèrent ESTEREL$^-$ dans le cadre des algèbres de processus.

Un programme ESTEREL$^-$ réagit à des signaux externes en engendrant d'autres signaux: l'exécution d'un programme ESTEREL$^-$ P est une suite de réactions :

$$
\begin{array}{rcccccccc}
 & & I_0 & & I_1 & \cdots & I_i & & \cdots \\
 & & | & & | & & | & & \\
[\![P]\!] & : & \circ & — & \circ & \cdots & \circ & — & \cdots \\
 & & | & & | & & | & & \\
 & & O_0 & & O_1 & \cdots & O_i & & \cdots
\end{array}
$$

où $I_1, \cdots, I_n$ sont des ensembles de signaux externes. $O_1, \cdots, O_n$ sont des ensembles de signaux émis par le programme et l'entier i rythme les différents instants. Rappelons que le langage étant synchrone, les signaux de I_i et de O_i sont présents en *même temps*.

3.1 Réaction et évènement

Une *réaction* d'un programme ESTEREL$^-$, est une fonction qui détermine les signaux émis par le programme en fonction des signaux d'entrée. On appelle *évènement*, un ensemble de signaux correspondant à un contexte dans lequel s'exécute un programme ESTEREL$^-$: l'ensemble des signaux présents simultanément dans une réaction. Nous appellerons *évènement d'entrée* d'une réaction, le sous-ensemble des signaux externes présents dans une réaction.

3.2 Environnement et formalisme des règles de comportement

Nous emprunterons le style des règles comportementales de Plotkin [11] pour représenter le comportement d'un programme ESTEREL$^-$. Nous utiliserons plus particulièrement le formalisme introduit par Kahn, dans [9], pour décrire

les pas atomiques de réactions. Un programme ESTEREL$^-$ est alors équivalent à une machine de Mealy ([8]) dont les langages d'entrée et de sortie sont les ensembles des signaux respectivement reçus et émis (l'ensemble des signaux d'entrée étant inclus dans l'ensemble des signaux de sortie). Si le programme ESTEREL$^-$ est sémantiquement incorrect, nous ne construirons pas cette machine.

Avant de décrire formellement les structures utilisées dans notre sémantique, donnons la forme générale de notre sémantique. Les règles de comportement sont de la forme :

$$(E,\ \rho) \vdash STAT \overset{(\mathcal{S},n)}{\Longrightarrow} STAT'$$

où :

- E est l'environnement d'évaluation,
- ρ est la relation de causalité,
- $STAT$ et $STAT'$ sont des instructions ESTEREL$^-$,
- $\mathcal{S} \subset E$ est l'ensemble des signaux émis par l'instruction $STAT$,
- n est le niveau de terminaison.

Cette notation exprime que globalement l'instruction $STAT$ peut réagir dans l'environnement E se transforme alors en $STAT'$.

Environnement

Nous noterons S l'ensemble des signaux ESTEREL$^-$. On appelle *état* du signal $s \in S$ un élément de l'ensemble $\{s^+,\ s^-\}$. Cet élément représente respectivement la présence ou l'absence du signal s. Nous appelerons *environnement* à l'instant t l'ensemble E_t des états des signaux intervenant dans un module à un instant t. Un environnement permet de déterminer si un signal est présent ou absent dans une réaction. S'il n'y a pas d'ambiguité sur l'instant t, on parlera alors simplement d'environnement. La sémantique que nous allons décrire, permettra de trouver les environnements de réaction qui contiennent l'environnement d'entrée et les signaux émis dans la réaction. Un évènement peut se décrire comme un sous ensemble d'un environnement: c'est l'ensemble des signaux présents dans l'environnement.

Pour pouvoir manipuler un environnement, on se donne une fonction partielle commutative, notée $\uplus : E \times E \rightharpoonup E$, qui permet de combiner deux environnements. Cette fonction est partielle, afin de respecter la validité des environnements en ne définissant que des environnements cohérents. De façon formelle $\uplus$ est définie si $s^* = s^+$ ou s^-, $s^{\overline{-}} = s^+$ et $s^{\overline{+}} = s^-$ par :

$$F \uplus G \begin{cases} \text{non défini ssi } \exists s^{*} \in F \text{ et } s^{\overline{*}} \in G \\ F \cup G \text{ sinon} \end{cases}$$

Par exemple :

$$\{s_1^+,\ s_3^-\} \uplus \{s_2^+\} = \{s_1^+,\ s_2^+,\ s_3^-\}$$

$$\{s_1^+,\ s_2^+\} \uplus \{s_2^-\} \text{ est indéfini}$$

$$\{s_1^+,\ s_2^+,\ s_3^-\} \uplus \{s_2^+\} = \{s_1^+,\ s_2^+,\ s_3^-\}$$

Ordre partiel

Certains opérateurs (comme **present**) imposent des conditions de dépendance entre signaux. Ainsi la présence d'un signal peut dépendre de la présence d'autres signaux. Si la présence de l'un de ceux-ci est elle-même dépendante de la présence du premier signal considéré, on a ce que l'on appelle un *cycle de causalité*. (ex. **present** s **then emit** t | **present** t **then emit** s la présence du signal t dépend de la présence du signal s qui elle-même dépends de la présence de t). Ces cycles de causalité peuvent rendre les réactions non-déterministes (cas précédent où s et t peuvent être tous les deux présents ou absents). Il nous faut donc refuser des programmes ayant de tels cycles.

Pour cela on associe à chaque environnement un ordre partiel strict[1] sur les noms de signaux qui symbolise cette inter-dépendance. Elle permet de vérifier la validité sémantique associée à la notion de causalité en refusant les cycles (par anti-symétrie de la relation). Nous l'appellerons *relation de causalité*. Elle sera déduite par les règles de comportements comme pour l'environnement. Par exemple, soit l'expression **present** s **then emit** t, l'émission du signal t dépend de la présence du signal s, ce qu'on dénotera par $s \prec t$. La relation devant être anti-symétrique, on refute ainsi les cycles de causalité entre signaux : **present** s **then emit** t || **present** t **then emit** s donnerait $s \prec t$ et $t \prec s$. On considèrera cette relation comme la fermeture transitive engendrée par un ensemble de couples que nous appellerons *dépendances*.

On définit alors une opération partielle d'union entre dépendances (comme pour l'environnement). Cette union (que nous noterons aussi $\uplus$) n'est définie que quand l'union ensembliste des dépendances génère une relation d'ordre stricte par fermeture transitive. Soit ρ' la fermeture transitive de $\rho_1 \cup \rho_2$. On définit $\rho_1 \uplus \rho_2$ par :

$$\rho_1 \uplus \rho_2 \begin{cases} \text{non défini ssi } \exists s \in \rho' \text{ tel que } s \prec s \\ \rho_1 \cup \rho_2 \text{ sinon} \end{cases}$$

[1] Les règles opérationelles permettent de considérer un ordre non strict, mais cela n'apporte rien à la sémantique des programmes ESTEREL. Par cohérence nous imposerons que l'ordre soit strict.

3.2.1 Niveaux de terminaison

Pour pouvoir faire dépendre la réaction d'un programme de son passé, ESTEREL a adopté une structure impérative de *séquencement* d'actions. Cela introduit dans ESTEREL la notion de *terminaison*. La terminaison normale, le non passage en séquence et le mécanisme d'exception de ESTEREL définissent différents niveaux de terminaison. L'opérateur de séquence permet d'absorber les terminaisons de passage en séquence, en commençant à exécuter la seconde instruction *simultanément* avec la terminaison de la première. Il inhibe ainsi la marque de terminaison de la première instruction et permet de coupler plusieurs instructions dans le même instant. Nous représenterons ces différents niveaux de terminaison par un entier, comme dans [4].

3.3 les règles de comportement des instructions

Les réactions d'un programme ESTEREL$^-$ sont définies en fonctions des signaux d'entrée. Dans notre formalisme, donner une sémantique à une expression ESTEREL$^-$ revient à trouver un environnement d'évaluation et une relation d'ordre entre les signaux de la réaction, valides pour les règles des opérateurs du langage définies ci après. Chaque contrainte sémantique (cohérence des signaux présents ou absents, présence d'un signal local uniquement s'il est émis, ...) s'exprime par l'impossibilité de trouver (à cause de la notion de définition partielle) un environnement ou une relation d'ordre associée à cette réaction. Ainsi si un environnement et une relation peuvent être synthétisés pour valider une réaction, alors celle-ci sera cohérente.

Contrairement à l'usage habituel, cette sémantique synthétise un environnement au lieu de le considérer comme un attribut hérité (d'un ensemble de déclarations par exemple).

Les règles de sémantique sont pour chaque opérateurs de ESTEREL$^-$:

- **nothing** : la terminaison en séquence se représente par une terminaison de niveau 0.

$$(\emptyset.\emptyset) \vdash \texttt{nothing} \overset{(\emptyset.\,0)}{\Longrightarrow} \texttt{nothing}$$

- **halt** : la terminaison de bloquage se traduit par une terminaison de niveau 1.

$$(\emptyset.\emptyset) \vdash \texttt{halt} \overset{(\emptyset.\,1)}{\Longrightarrow} \texttt{halt}$$

- **emit** : l'émission d'un signal se traduit par la présence de ce signal et par un passage en séquence. Il peut sembler redondant de mettre le signal dans la liste des signaux émis alors qu'on l'ajoute à l'environnement. Cela permet de tester qu'un signal local ne sera présent que s'il a été émis.

$$(\{s^+\}, \emptyset) \vdash \texttt{emit(s)} \overset{(\{s\},\, 0)}{\Longrightarrow} \texttt{nothing}$$

Il est important de noter que $\uplus$ est plus qu'un simple opérateur de construction de l'environnement (par synthèse) mais qu'il vérifie en meme temps la cohérence de la structure créée.

- **sequence** : le passage en séquence s'effectue en fonction du niveau de terminaison de l'instruction précédente.

$$\frac{(\check{E}_1, \rho_1) \vdash stat_1 \overset{(\mathcal{S}_1,\, 0)}{\Longrightarrow} stat'_1.\ (\check{E}_2, \rho_2) \vdash stat_2 \overset{(\mathcal{S}_2,\, n)}{\Longrightarrow} stat'_2}{(\check{E}_1 \uplus \check{E}_2,\ \rho_1 \uplus \rho_2) \vdash stat_1 ; stat_2 \overset{(\mathcal{S}_1 \cup \mathcal{S}_2,\, n)}{\Longrightarrow} stat'_2}$$

$$\frac{(E, \rho) \vdash stat_1 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'_1.\ n > 0}{(E, \rho) \vdash stat_1 ; stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'_1 ; stat_2}$$

- **loop** : l'instruction d'itération consiste à développer le comportement.

$$\frac{(E, \rho) \vdash stat\texttt{:loop}\ stat\ \texttt{end} \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'}{(E, \rho) \vdash \texttt{loop}\ stat\ \texttt{end} \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'}$$

Cette règle comporte un point fixe dont la solution n'est pas toujours définissable. Toutefois avec la contrainte (vérifiable statiquement) que les boucles ne terminent pas instantanément. cette équation admet toujours une solution.

- **present** : la présence d'un signal est déterminée par l'état de celui-ci dans l'environnement d'évaluation.

$$\frac{(E, \rho) \vdash stat_1 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'_1}{(E \uplus s^+,\ \rho \uplus (s^+,\ \mathcal{S})) \vdash \texttt{present}\ s\ \texttt{then}\ stat_1\ \texttt{else}\ stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'_1}$$

$$\frac{(E,\rho) \vdash stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat_2'}{(E \uplus s^-,\ \rho \uplus (s^-,\ \mathcal{S})) \vdash \texttt{present}\ s\ \texttt{then}\ stat_1\ \texttt{else}\ stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat_2'}$$

la notation $\rho \uplus (s,\ \mathcal{S})$ est une extension de l'union de deux dépendances. elle se comprend par $(s,\ \mathcal{S}) = \{(s, x) | x \in \mathcal{S}\}$.

- **watching** : attendre un signal consiste à ne réaliser la réaction que si le signal est présent dans l'environnement.

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, 0)}{\Longrightarrow} stat'}{(E,\rho) \vdash \texttt{do}\ stat\ \texttt{watching}\ s \overset{(\mathcal{S},\, 0)}{\Longrightarrow} \texttt{nothing}}$$

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat',\ n > 0}{(E,\rho) \vdash \texttt{do}\ stat\ \texttt{watching}\ s \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{present}\ s\ \texttt{else do}\ stat'\ \texttt{watching}\ s}$$

- **parallel** : le niveau de terminaison du parallèle est le maximum des niveaux de terminaison des branches parallèles.

$$\frac{(\check{E}_1,\ \rho_1) \vdash stat_1 \overset{(\mathcal{S}_1,\, n_1)}{\Longrightarrow} stat_1',\ (\check{E}_2,\ \rho_2) \vdash stat_2 \overset{(\mathcal{S}_2,\, n_2)}{\Longrightarrow} stat_2'}{(\check{E}_1 \uplus \check{E}_2,\ \rho_1 \uplus \rho_2) \vdash (stat_1 \| stat_2) \overset{(\mathcal{S}_1 \cup \mathcal{S}_2,\ max(n_1,\ n_2))}{\Longrightarrow} (stat_1' \| stat_2')}$$

On peut remarquer la similitude avec la règle de séquence. A la terminaison près, faire deux instructions en séquence ou en parallèle est équivalent.

- **test** : vérifier une condition est équivalent à vérifier la présence d'un signal représentant la valeur booléene de cette condition.

$$\frac{(E,\rho) \vdash stat_1 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat_1'}{(E \uplus c^+,\ \rho) \vdash \texttt{if}\ c\ \texttt{then}\ stat_1\ \texttt{else}\ stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat_1'}$$

$$\frac{(E,\rho) \vdash stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat_2'}{(E \uplus c^-,\ \rho) \vdash \texttt{if}\ c\ \texttt{then}\ stat_1\ \texttt{else}\ stat_2 \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat_2'}$$

- **trap** : chaque récupération d'échappement diminue le niveau de celui-ci d'une unité, jusqu'à être récupéré pour passer en séquence.

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat',\ n < 2}{(E,\rho) \vdash \texttt{trap in } stat \texttt{ end} \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{nothing}}$$

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat',\ n = 2}{(E,\rho) \vdash \texttt{trap in } stat \texttt{ end} \overset{(\mathcal{S},\, 0)}{\Longrightarrow} \texttt{nothing}}$$

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat',\ \text{and } n > 2}{(E,\rho) \vdash \texttt{trap in } stat \texttt{ end} \overset{(\mathcal{S},\, n-1)}{\Longrightarrow} \texttt{trap in } stat' \texttt{ end}}$$

- **exit** : le niveau de terminaison de l'instant est relatif à celui de l'échappement.

$$\frac{}{(\emptyset,\emptyset) \vdash \texttt{exit } n \overset{(\mathcal{S},\, n+1)}{\Longrightarrow} \texttt{nothing}}$$

- **local** : pour vérifier la cohérence d'un signal local, il suffit de vérifier que si le signal est présent dans la réaction c'est parce qu'il a été émis dans l'instant. La visibilité du signal est alors définie par les opérations $/s$ définies de façon classique par soustraction ensembliste :

$$E/s = \{t^{\star} | t^{\star} \in E \text{ et } t \neq s\}$$

$$\rho/s = \{(x,y) | (x,y) \in \rho \text{ et } x \neq s^{\star},\ y \neq s^{\star}\}$$

$$\frac{(E \uplus \{s^{+}\},\ \rho) \vdash stat \overset{(\mathcal{S} \cup \{s\},\, n)}{\Longrightarrow} stat'}{(E/s,\ \rho/s) \vdash \texttt{local } s \texttt{ in } stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{local } s \texttt{ in } stat}$$

$$\frac{(E \uplus \{s^{-}\},\ \rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'}{(E/s,\ \rho/s) \vdash \texttt{local } s \texttt{ in } stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{local } s \texttt{ in } stat}$$

- **input** : tout comportement contenant un signal externe est valide si ce comportement n'impose pas de condition sur la présence ou l'absence de ce signal, ou qu'il existe bien deux comportements pour la présence et l'absence de ce signal.

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat' \text{ and } s^+ \notin \check{E} \text{ and } s^- \notin \check{E}}{(E,\rho) \vdash \texttt{input}\ s\ \texttt{in}\ stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{input}\ s\ \texttt{in}\ stat'}$$

$$\frac{\begin{array}{l}(\check{E}_1,\ \rho_1) \vdash stat \overset{(\mathcal{S}_1,\, n_1)}{\Longrightarrow} stat' \text{ and } s^+ \in \check{E}_1, \text{ and} \\ (\check{E}_2,\ \rho_2) \vdash stat \overset{(\mathcal{S}_2,\, n_2)}{\Longrightarrow} stat' \text{ and } s^- \in \check{E}_2\end{array}}{\begin{array}{l}(\check{E}_1,\ \rho_1) \vdash \texttt{input}\ s\ \texttt{in}\ stat \overset{(\mathcal{S}_1,\, n_1)}{\Longrightarrow} \texttt{input}\ s\ \texttt{in}\ stat' \text{ and} \\ (\check{E}_2,\ \rho_2) \vdash \texttt{input}\ s\ \texttt{in}\ stat \overset{(\mathcal{S}_2,\, n_2)}{\Longrightarrow} \texttt{input}\ s\ \texttt{in}\ stat'\end{array}}$$

Une conséquence de ces règles est qu'un signal déclaré **input** ne peut être émis dans la réaction.

- **output** : il n'y a aucune contrainte sur un signal déclaré **output**.

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat'}{(E,\rho) \vdash \texttt{output}\ s\ \texttt{in}\ stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{output}\ s\ \texttt{in}\ stat'}$$

Un signal **output** d'ESTEREL$^-$ se comporte en fait comme un signal **input-output** d'ESTEREL.

- **module** : cette instruction permet de compiler séparément des modules ESTEREL$^-$. Ceux-ci ne doivent pas avoir des exceptions non récupérées.

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},\, n)}{\Longrightarrow} stat' \text{ and } n < 2}{(E,\rho) \vdash \texttt{module}\ (ident, stat) \overset{(\mathcal{S},\, n)}{\Longrightarrow} \texttt{module}\ (ident, stat')}$$

- **copymodule** : on renomme les signaux émis et d'entrés. Les renommages que nous considèrons ne doivent pas changer le type des signaux (input et output). Le seul cas que nous pourions considérer est le renommage d'un signal d'entrée en un signal de sortie (en effet, il n'y a pas de contrainte sur un signal output), mais ceci n'apporte rien car un signal input ne peut

être émis s'il est absent. On notera r toute fonction de renommage alphabetique et $E[r]$ (resp. $\rho[r]$) l'environnement (resp. la relation) renommé par r.

$$\frac{(E,\rho) \vdash \texttt{module}\ (id, stat) \overset{(\mathcal{S},n)}{\Longrightarrow} \texttt{module}\ (id, stat')}{(E[r],\ \rho[r]) \vdash \texttt{copymodule}\ (id, stat, r) \overset{(\mathcal{S}[r],n)}{\Longrightarrow} \texttt{copymodule}\ (id, stat', r)}$$

- **affaiblissement** : les environnements que nous avons considérés sont minimaux pour obtenir une réaction. Tout autre description de signal non utilisée dans un module est insignifiante pour la réaction. Cette propriété s'exprime par :

$$\frac{(E,\rho) \vdash stat \overset{(\mathcal{S},n)}{\Longrightarrow} stat'}{(E \uplus E',\ \rho) \vdash stat \overset{(\mathcal{S},n)}{\Longrightarrow} stat'}$$

cette règle représente l'affaiblissement de l'opérateur $\uplus$ par rapport à $\vdash$.

Remarque : Il est important de rappeller que cette sémantique ne vérifie pas la détection des boucles instantanées.

3.4 Execution et déterminisme

L'exécution d'un programme ESTEREL$^-$ sur une machine séquentielle est alors une séquence quelconque d'émissions des signaux de sortie. L'environnement d'évaluation de la réaction étant le plus petit environnement calculable contenant les signaux d'entrée.

Afin de pouvoir démontrer que le comportement de toute expression est déterministe, il nous faut d'abords établir quelques résultats.

Lemme 1 *Tout arbre de preuve d'un comportement qui suppose présent un signal local s contient une occurence de la règle* `emit` *appliquée au signal s.*

Preuve. Les règles du `local` imposent que si le signal est présent dans l'environnement alors il doit être émit. Le seul moyen d'émettre un signal est donné par l'opérateur `emit`.

Lemme 2 *Un terme T quelconque admettant deux arbres de preuve de comportements, distincts pour un même environnement d'entrée, admet un signal local s et une occurence u telle que $T|u =$* `present` *$s\ldots$ tel que pour toute occurence distincte des arbres il exite un chemin passant par u.*

Preuve. Les seuls opérateurs qui permettent d'avoir des sous-arbres de preuve distincts sont **present** et **if**. L'opérateur **if** induit alors un environnement distincts d'entrée ce qui est contraire à l'hypothèse. Donc seul l'opérateur **present** appliqué à un signal local permet d'obtenir des sous-arbres distincts pour un même environnement d'entrée.

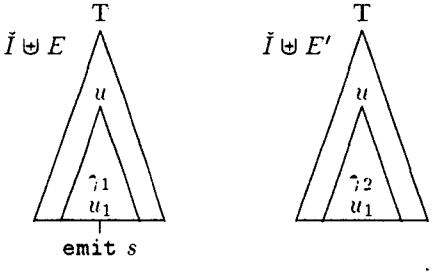
Lemme 3 *Si un terme admet deux comportements. distincts pour un même environnement d'entrée. obtenus en supposant présent ou absent un signal local alors il existe un autre signal local dont il dépend.*

Formellement. soit un terme T tel qu'il existe une occurence u avec $T|u =$ **local** *s. S'il existe*

$$\Gamma_1 \models \check{I} \uplus E \vdash T \overset{(S,n)}{\Longrightarrow} T_1 et\ \Gamma_2 \models \check{I} \uplus E' \vdash T \overset{(S',n')}{\Longrightarrow} T_1'$$

deux arbres de preuve de comportement tels que la règle $\mathtt{local}_1 \in \Gamma_1$ *et la règle* $\mathtt{local}_2 \in \Gamma_2$. *alors il existe un autre signal local t tel que $t \prec s$ et $t^* \in \Gamma_1$, $\bar{t^*} \in \Gamma_2$.*

Preuve. Considèrons les deux sous-arbres de preuve des deux comportements au noeud $T|u =$ **local** s. Notons γ_1 le sous-arbre ayant été obtenu en appliquant la première règle de l'opérateur **local** (qui considère le signal présent) et γ_2 le sous-arbre avec la seconde.



L'environnement synthétisé à la racine de γ_1 suppose s présent. Par le lemme 1 il existe u_1 occurence de γ_1 telle que $\gamma_1|u_1 =$ **emit** s et il n'existe pas une telle occurence dans γ_2. Les deux arbres sont donc distincts. Le lemme 2 permet d'affirmer qu'il existe un signal local t tel qu'il existe un noeud **present** t... entre la racine de γ_1 et u_1. On a alors $t \prec s$

On peut alors énoncer la propriété fondamentale vérifiée par cette sémantique :

Propriété 1 *Tous les comportements obtenus sont déterministes.*

Preuve. Le non déterminisme correspondrait à la possibilité de plusieurs réactions à un même environnement d'entrée. Cela supposerait donc qu'il existe un signal local s tel qu'il existe un comportement supposant que ce signal soit présent et un autre qui supposerait que ce signal soit absent. Par le lemme 3 cela suppose qu'il existe un autre signal local t qui vérifie les mêmes propriétés et tel que $t \prec s$. Et réitérant sur le signal t on obtient une suite infinie de dépendance entre signaux locaux. Ceci impliquerait un cycle de dépendances entre certains signaux locaux.

3.5 Extension à ESTEREL

Signaux valués

Pour pouvoir étendre cette sémantique au langage ESTEREL avec des valeurs sur les signaux, il faut étendre l'ensemble de description des signaux et considérer une relation de causalité plus complexe. A chaque signal s on associe toujours un couple $\{s^+, s^-\}$ mais aussi des descriptions des occurences d'émission de ce signal. Une occurrence d'un signal est décrite par $s_{occ}(v)$ où v représente la valeur locale du signal au moment de l'émission. Cette occurence est reliée à la dépendance entre signaux par :

$$s_{occ}(v) > s$$

La valeur du signal est alors calculée (par combinaison des valeurs locales) lors de l'exécution à l'émission du signal.

Relation de contrainte

Le langage ESTEREL permet de définir des contraintes sur les dépendances des signaux d'entrée. Ces contraintes peuvent facilement être représentées dans notre sémantique. Il suffit d'associer à chaque environnement un ensemble d'équations de la forme :

$$s^+ \Rightarrow t^-$$

L'environnement de description des signaux étant alors modifiés par application de ces équations. Ce modèle pourrait permettre une extension de la définition des relations aux signaux de sortie (comme validation d'assertion).

4 Exemples

Pour faciliter la compréhension de la sémantique de ESTEREL$^-$, nous allons donner quelques exemples de programmes et montrer comment on obtient leur sémantique comportementale. Pour cela, nous utiliserons un outil Prolog [5] qui nous permet d'évaluer les termes d'une algèbre dont on a introduit la sémantique opérationnelle sous forme de règles structurelles.

4.1 Premier exemple

Le premier exemple que nous considèrerons, est ce qu'on appelle un *dialogue instantané*. Cela consiste à émettre un signal local afin d'enclancher une réaction qui est décrite par un autre processus mis en parallèle. Ce signal simule un dialogue de validation entre les deux termes en parallèles. Montrons cet exemple directement à partir de notre environnement prolog. Il consiste à évaluer le terme suivant :

module TEST
    input INPUT:
    output OUTPUT:
    local s in
        present INPUT then emit s end || present s then emit OUPUT end
    end
end.

ce qui nous donne en prolog :

```
?- eval("emit s").
The behavior of "emit s" is :
  {(s),()} |-  -- ([s], 0) --> nothing
ok.

?- eval("present s then emit output end").
The behavior of "present s then emit output end" is :
  {(output,s),(s<output)} |-
      -- ([output], 0) --> nothing
  {(),(s<output)} |-
      -- ([], 0) --> nothing
ok.

?- eval("emit s||present s then emit output end").
The behavior of "emit s||present s then emit output end" is :
  {(output,s),(s<output)} |-
      -- ([output,s], 0) --> nothing||nothing
ok.

?- eval("present input then emit s end ||
         present s then emit output end").
The behavior of
   "present input then emit s end ||
    present s then emit output end"
is :
```

```
  {(input,output,s),(input<s,s<output)} |-
      -- ([output,s], 0) --> nothing||nothing
  {(output,s),(s<output)} |-
      -- ([output], 0) --> nothing||nothing
  |-  -- ([], 0) --> nothing||nothing
ok.

?-eval("module test input input in output output in
local s in present input then emit s end ||
present s then emit output end.").
The behavior of
   "module test ( input input in (output output in
    (local s in (present input then emit s end ||
    present s then emit output end))))"
is :
  {(input,output),(input<output)} |-
      -- ([output], 0) --> module ... (nothing||nothing)
  |-  -- ([], 0) --> module ... (nothing||nothing)
ok.
```

4.1.1 Second exemple

Le second exemple que nous considèrons est formé de termes qui comportent un cycle de causalité. Cet exemple montre comment la relation d'ordre induite sur les signaux. interdit tout comportement contenant un cycle de causalité.

```
?- eval("present s then emit t end").
The behavior of "present s then emit t end" is :
  {(s,t),(s<t)} |-
      -- ([t], 0) --> nothing
  |-  -- ([], 0) --> nothing
ok.

?- eval("present t then emit s end").
The behavior of "present t then emit s end" is :
  {(s,t),(t<s)} |-
      -- ([s], 0) --> nothing
  |-  -- ([], 0) --> nothing
ok.

?- eval("present s then emit t end||present t then emit s end").
The behavior of
```

```
   "present s then emit t end||present t then emit s end"
is :
ok.

?- eval("present s else  emit t end||present t else emit s end").
The behavior of
   "present s else emit t end||present t else emit s end"
is :
ok.

?- eval("local s in present s then emit s end").
The behavior of "local s in (present s then emit s end)" is :
ok.
```

5 Conclusion

Afin de mieux comprendre comment se situe cette sémantique par rapport à celles qui ont été définies par Gonthier [4] et Boussinot [3], montrons comment ces diverses sémantiques analysent les exemples suivants :

emit *s*: **present** *s* **else emit** *s* **end.** Ce programme est refusé par la sémantique des potentiels de [4] mais est accepté par celle de [3] et la nôtre.

present *s* **then** ... **end; emit** *s*. Ce programme est refusé par les sémantiques de [4] et de [3] mais est accepté par la nôtre.

present *s* **then emit** *s* **end.** Ce programme est refusé par les trois sémantiques. Toutefois, notre sémantique refuse ce programme uniquement pour des raisons de non-déterminisme potentiel.

Nous avons défini une sémantique d'ESTEREL de façon exhaustive. Il semble que des structures plus complexes (comme les structures d'évènements) permettent de synthétiser les différents comportements d'un terme de façon plus concise. Toutefois la structure d'ensembles de signaux partiellement ordonnés semble etre la structure minimale d'information qui permette la modularité. En cela, elle parait etre un bon candidat comme représentation commune pour la compilation séparée.

References

[1] D. Austry et G. Boudol.
"Algèbre de processus et synchronisation"
Theoretical Computer Science 30 (91-131). 1984.

[2] [BerCo84] G. Berry & L. Cosserat
The Synchronous Programming Language ESTEREL and its Mathematical Semantics.
"Seminar on Concurrency". Springler-Verlag LNCS 197 (1984).

[3] [Bo86] F. Boussinot
"Une sémantique du langage ESTEREL". Rapport INRIA. RR 577. 1986.

[4] [Gon88] G. Gonthier
Sémantique et modèles d'exécution des langages réactifs synchrones: application à ESTEREL. Thèse d'informatique. Université d'Orsay (1988).

[5] [Dou89] G. Doumenc
internal report. to appear.

[6] I.S.O.
Information processing - Open System Interconnection
The definition of the specification language LOTOS
Draft proposal. ISO/DP 8807 97/21 N 423 (1985).

[7] [HaPnu] D. Harel & A. Pnueli
"On the Development of Reactive Systems : Logic and Models of Concurrent Systems".
Proc. NATO Advanced Study Institute on Logics and Models for Verification and Specification of Concurrent Systems. NATO ASI Series F. vol. 13. Springler-Verlag. pp. 477-498 (1985).

[8] [HoU179] J. Hopcroft & J. Ullman
"Introduction to Automata Theory. Languages. and Computation"
Addison-Wesley. 1979.

[9] [Kh87] G. Kahn
"Natural Semantics"
Rapport INRIA. RR 601. 1987.

[10] [Mi80] R. Milner.
A Calculus for Communicating System
Lectures Notes in Computer Science 92. (1980).

[11] [Plo81] G. Plotkin
"A Structural Approach to Operational Semantics"

DAIMI FN-19. Computer Science Departement, Aarhus University. Denmark. 1981.

[12] [Ta85] J.M. Tanzi
"Traduction structurelle des programmes esterel en automate",
Thèse de docteur ingénieur,
Novembre 1985 - Université de Nice.

Imprimé en France
par
l'Institut National de Recherche en Informatique et en Automatique